# Sample Based Motion Planning Methods

Probablistic Road Maps

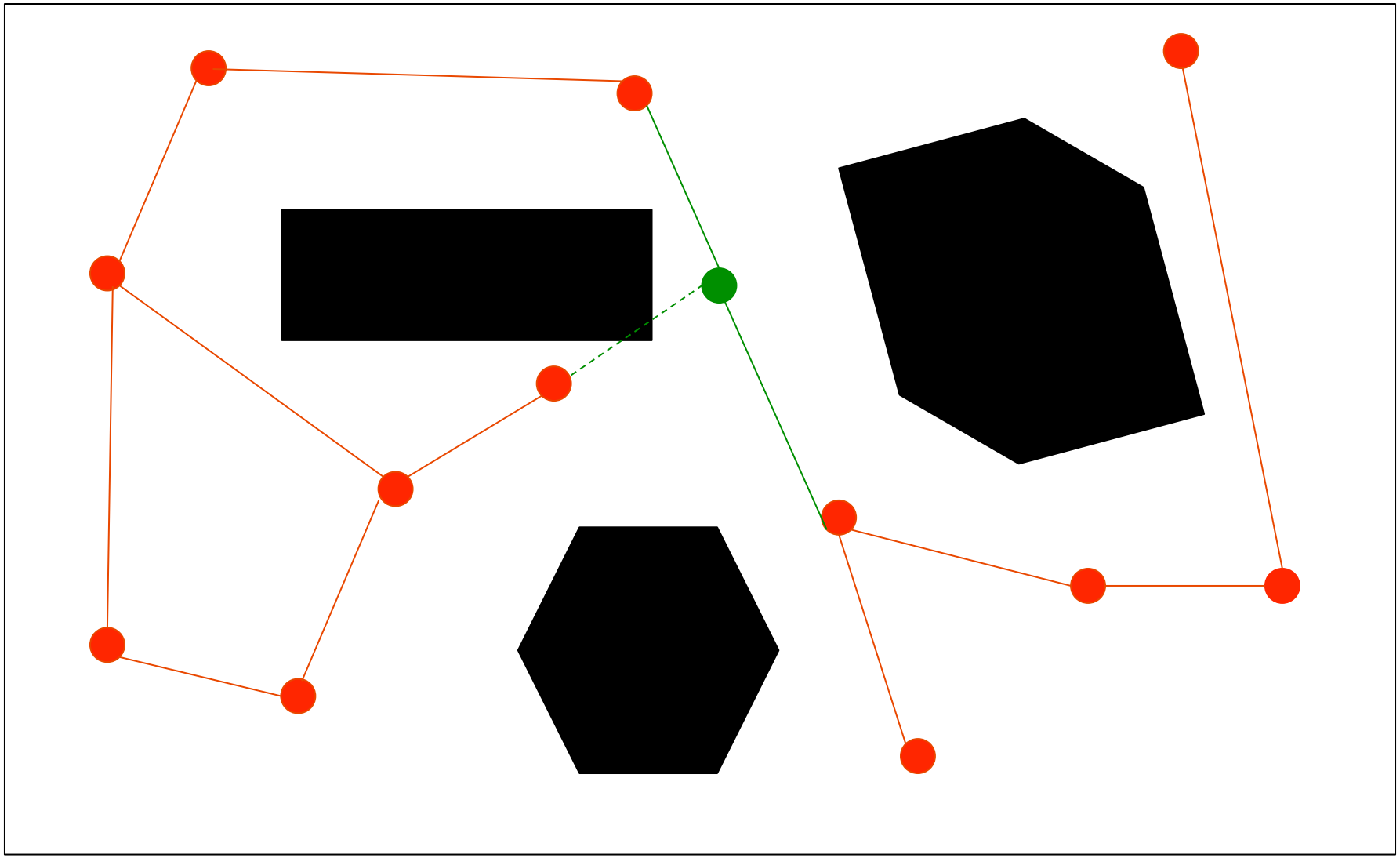
# SECTION 3.1

Penn

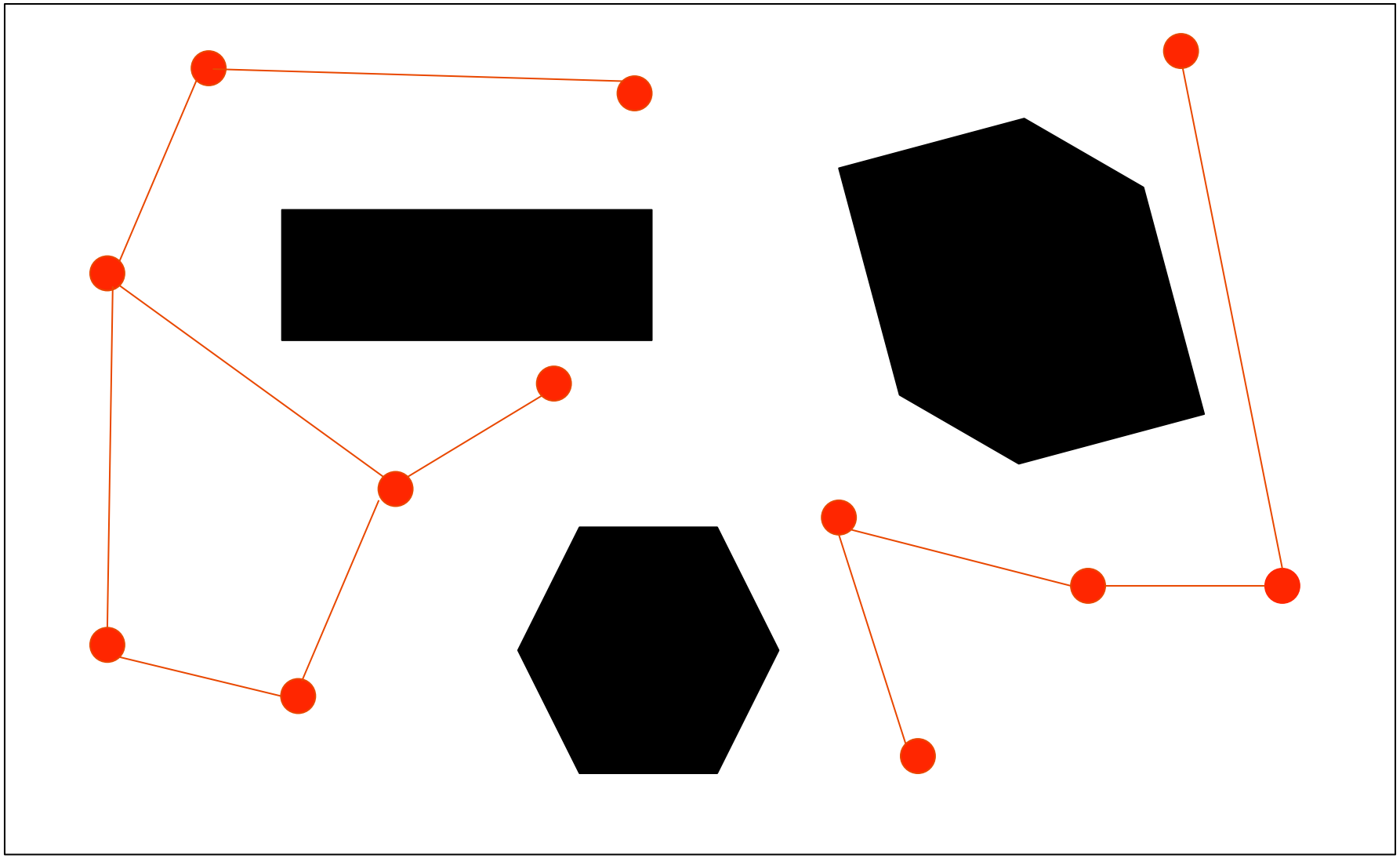
# Probablistic Road Map Pseudocode

- Repeat n times
  - Generate a random point in configuration space, **x**
  - If **x** is in freespace
    - Find the k closest points in the roadmap to **x** according to the Dist function
    - Try to connect the new random sample to each of the k neighbors using the LocalPlanner procedure. Each successful connection forms a new edge in the graph.

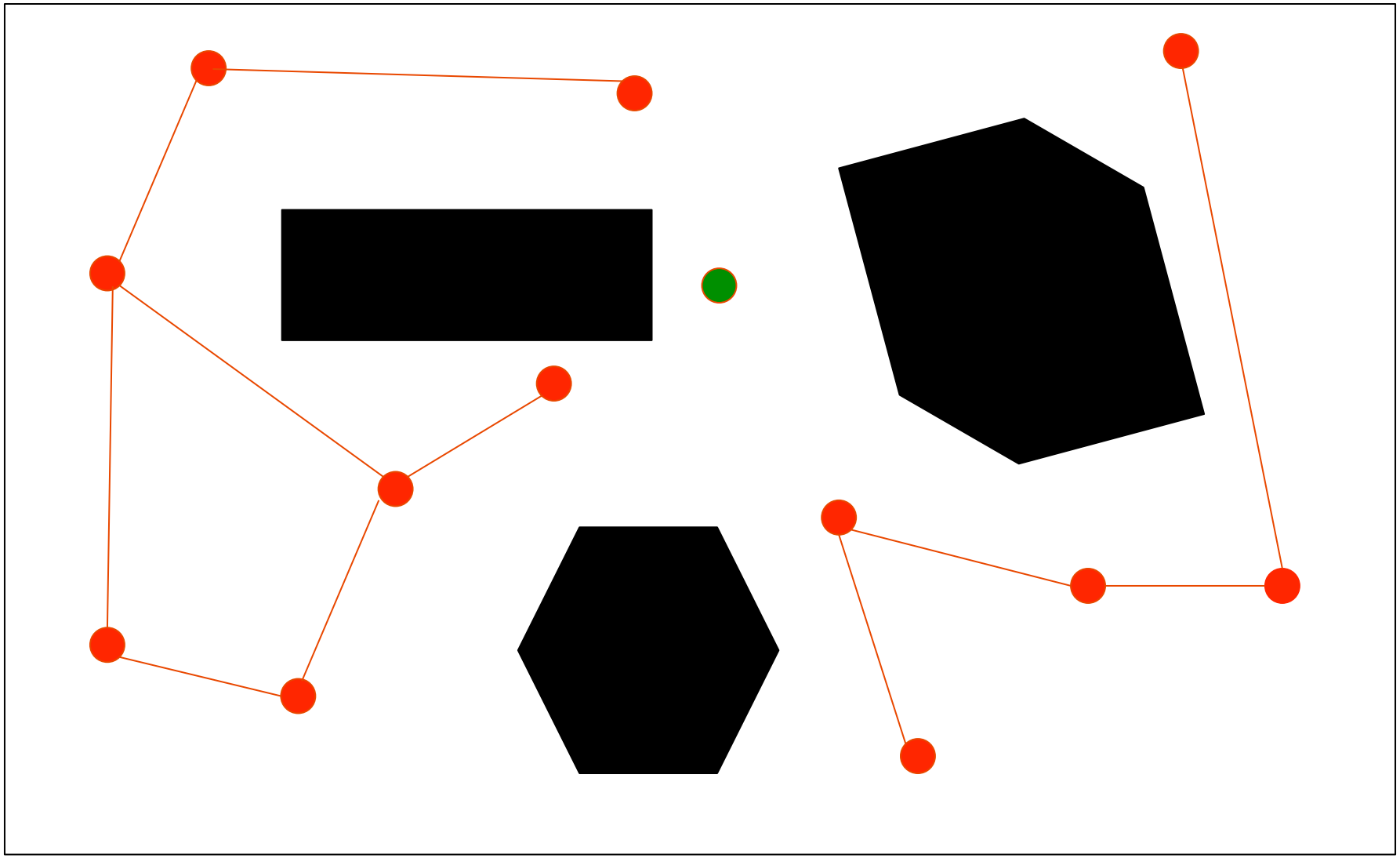
# Random Graph Construction



# Random Graph Construction



# Random Graph Construction



# The Dist function

- The PRM procedure relies upon a distance function, Dist,that can be used to gauge the distance between two points in configuration space. This function takes as input the coordinates of the two points and returns a real number:

$$Dist(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$$

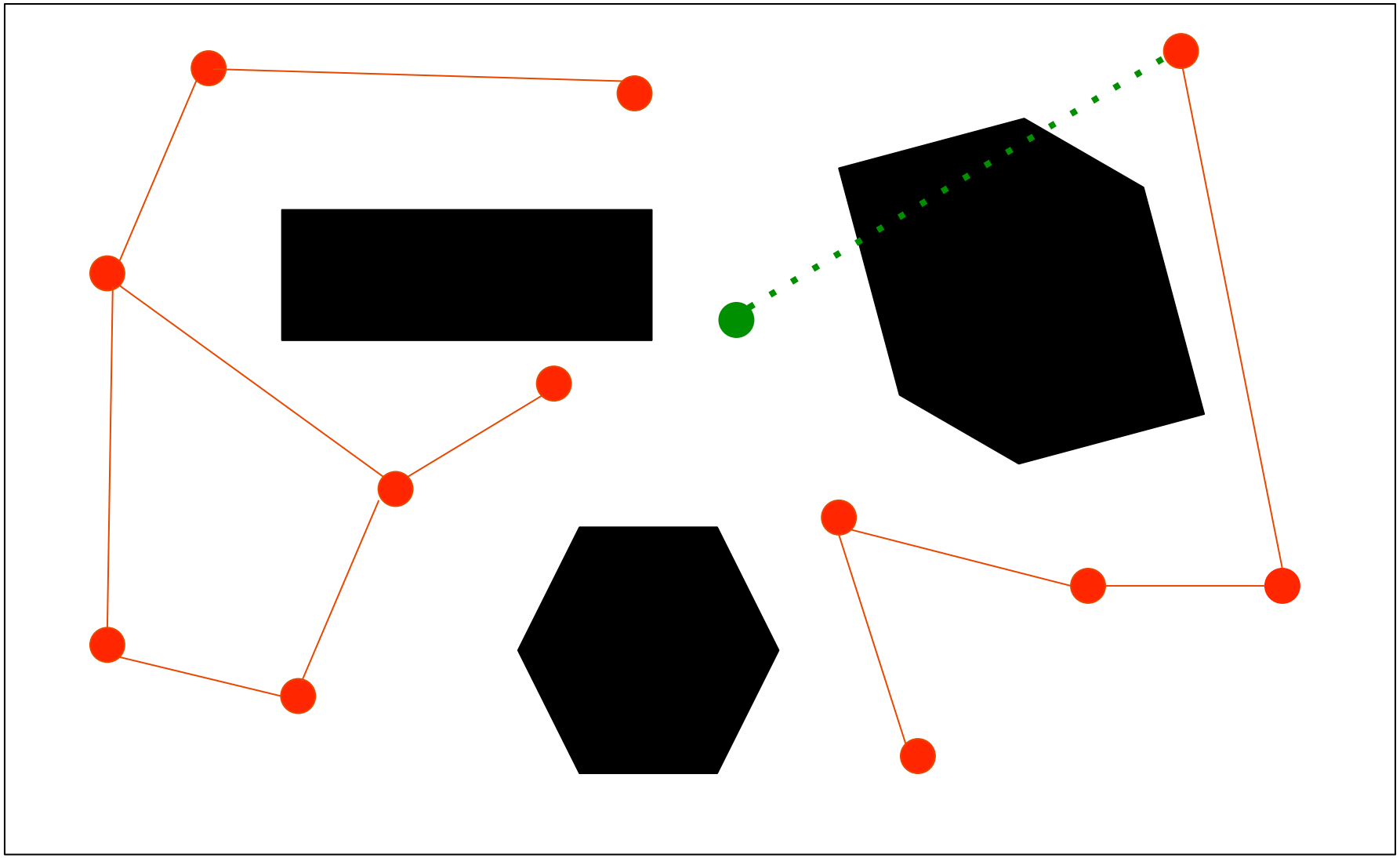
- Common choices for distance functions include:
  - The L1 distance : $Dist_1 = \sum_i |\mathbf{x}_i - \mathbf{y}_i|$
  - The L2 distance : $Dist_2 = \sqrt{(\sum_i (\mathbf{x}_i - \mathbf{y}_i)^2)}$

# Handling angular displacements

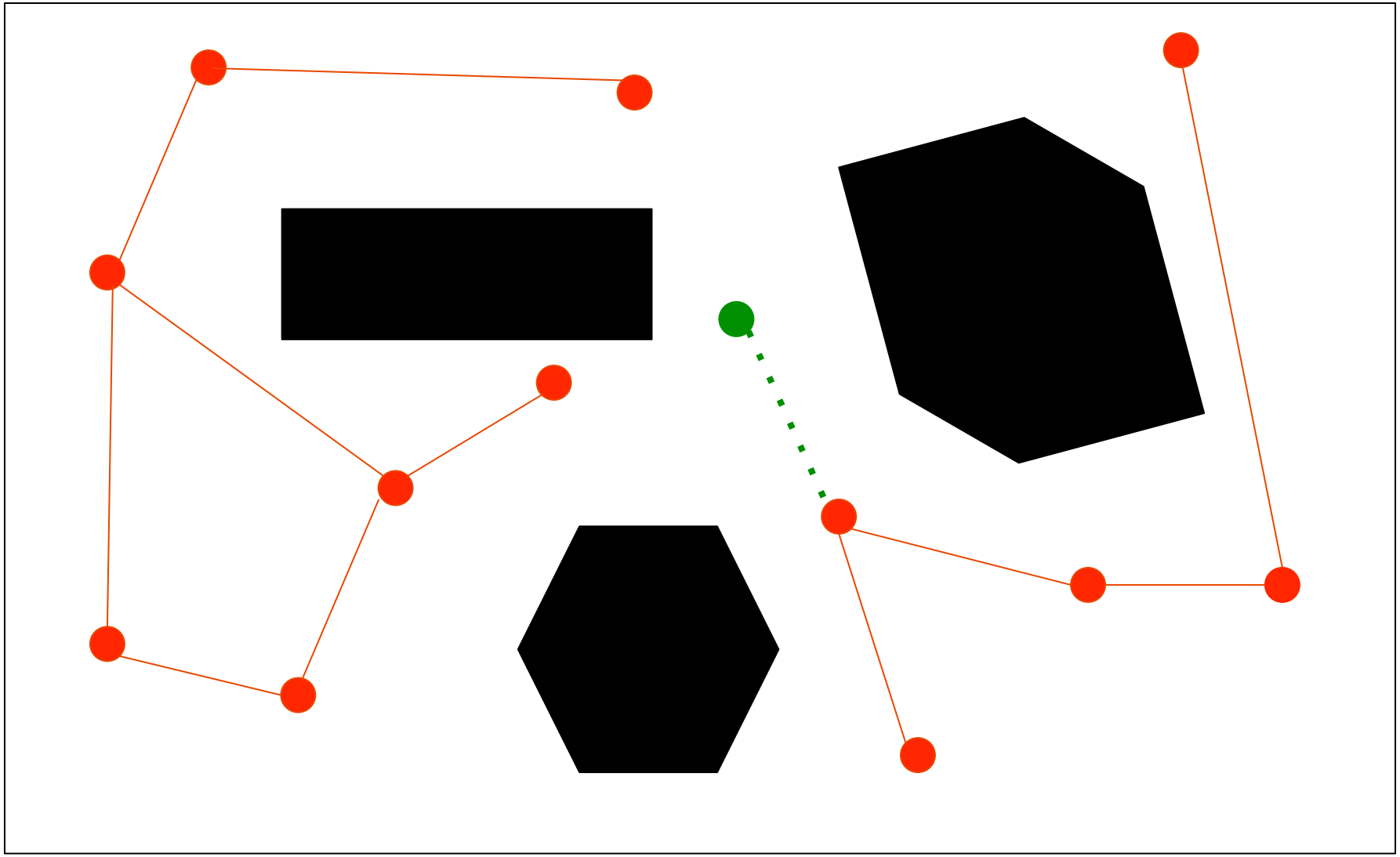
- There are often cases where some of the coordinates of the configuration space correspond to angular rotations. In these situations care must be taken to ensure that the *Dist* function correctly reflects distances in the presence of wraparound.
- For example if $\theta_1$ and $\theta_2$ denote two angles between 0 and 360 degrees the expression below can be used to capture the angular displacement between them.

$$Dist(\theta_1, \theta_2) = \min(|\theta_1 - \theta_2|, (360 - |\theta_1 - \theta_2|)) \quad (1)$$

# Checking for collision along a path

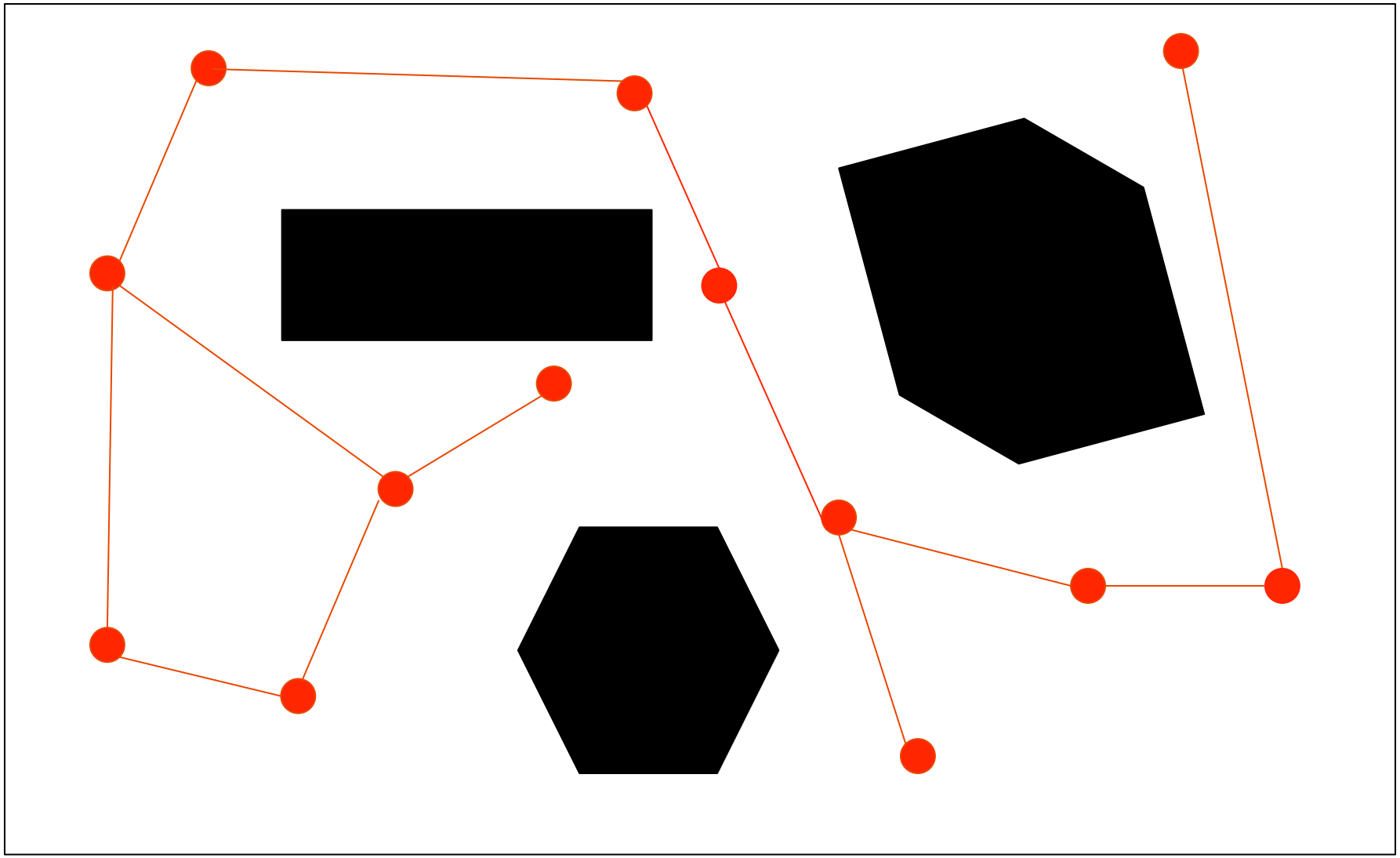


# Checking for collision along a path

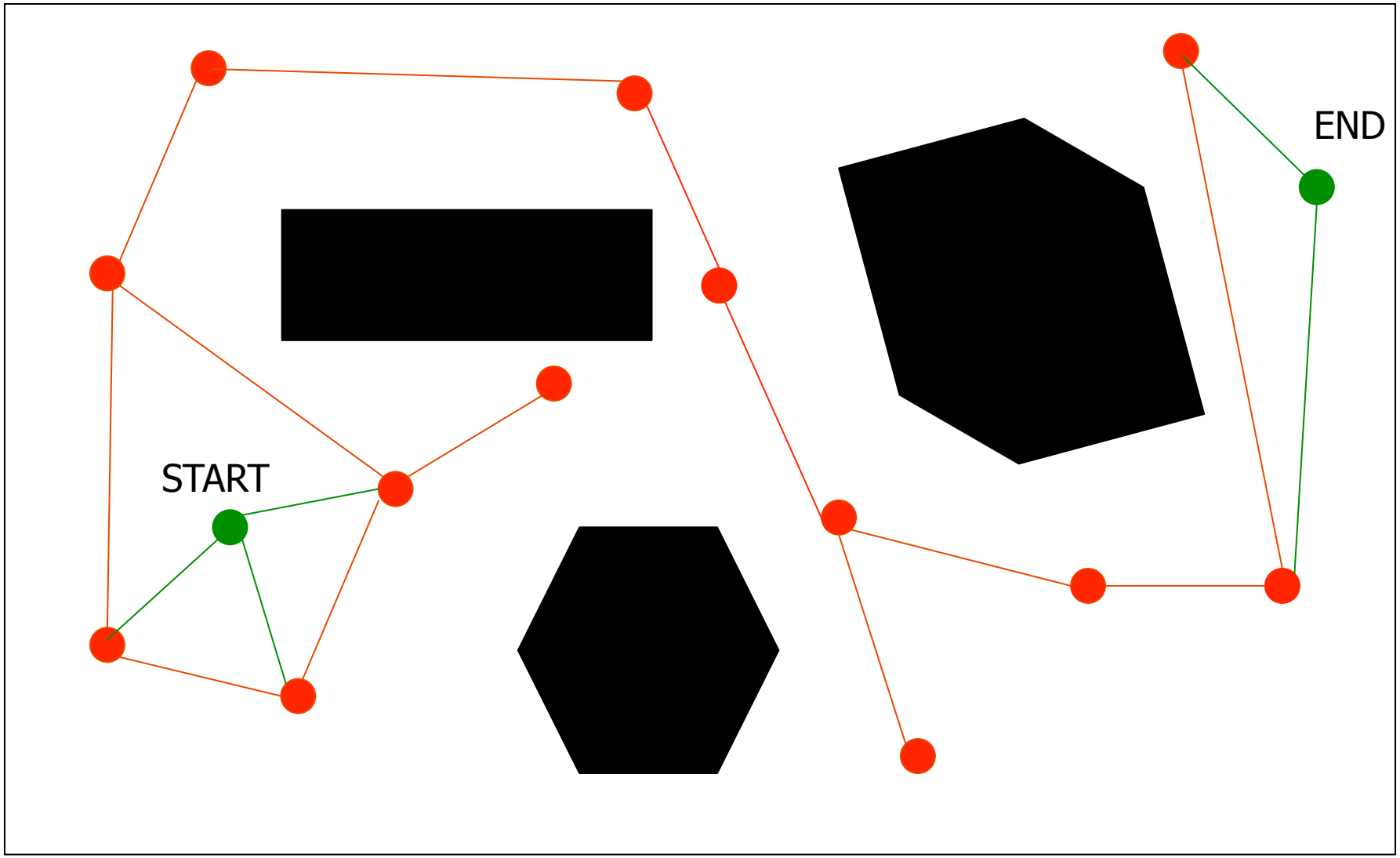


- This first slide shows the original Probabilistic roadmap constructed via random sampling.
- This second slide shows the roadmap augmented with the start and end nodes which are attached to the roadmap using the green edges
- This final version shows the path planned through the augmented graph

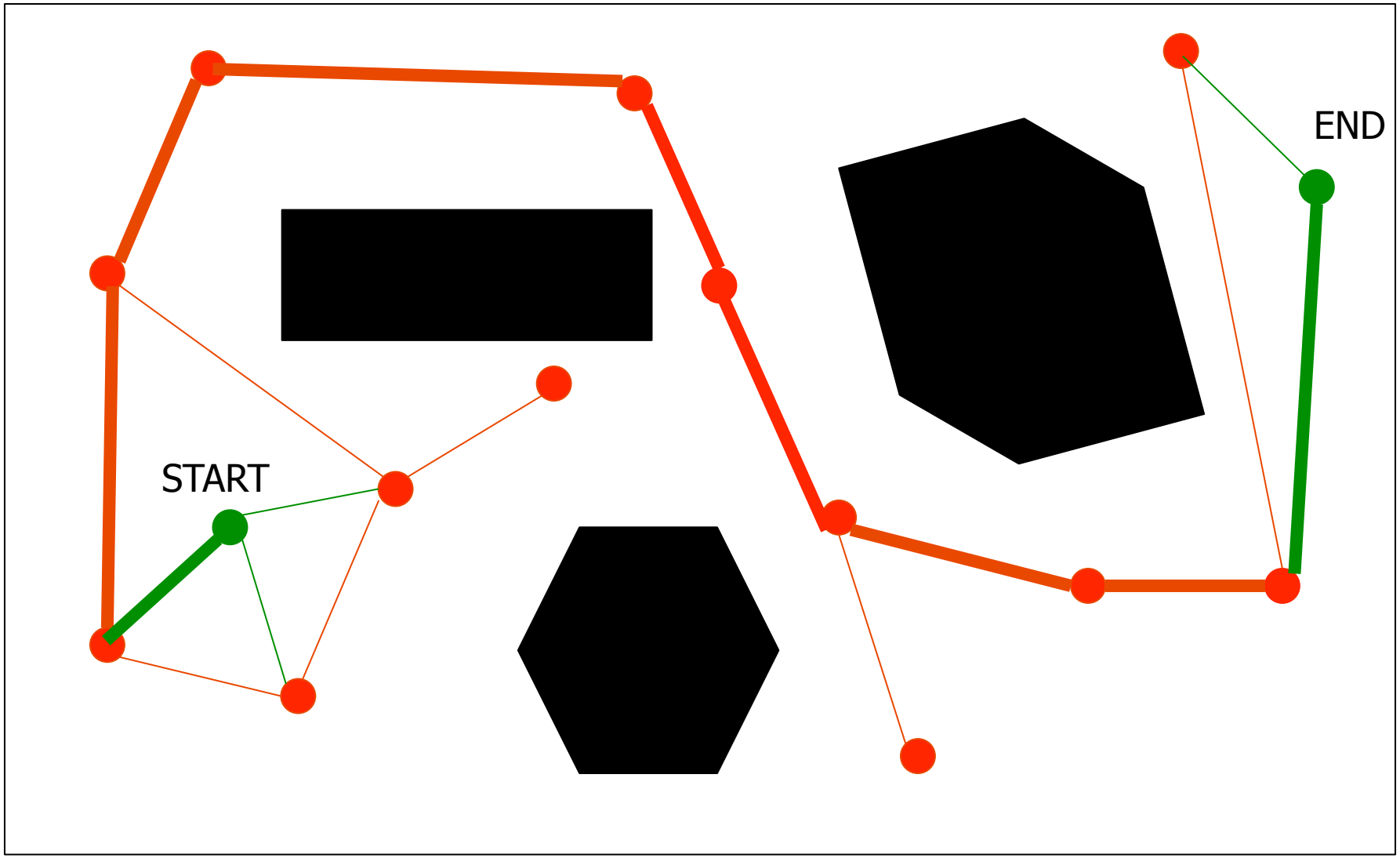
# Initial Road Map



# Road Map with Start and End added



# Final Route



- **Note that once the roadmap has been constructed it can be used to answer various path planning problems so the cost of constructing the roadmap graph can be amortized over multiple queries. This is great if you are going to be running the your robot back and forth through the same environment.**

Characteristics of Sample Based Planners

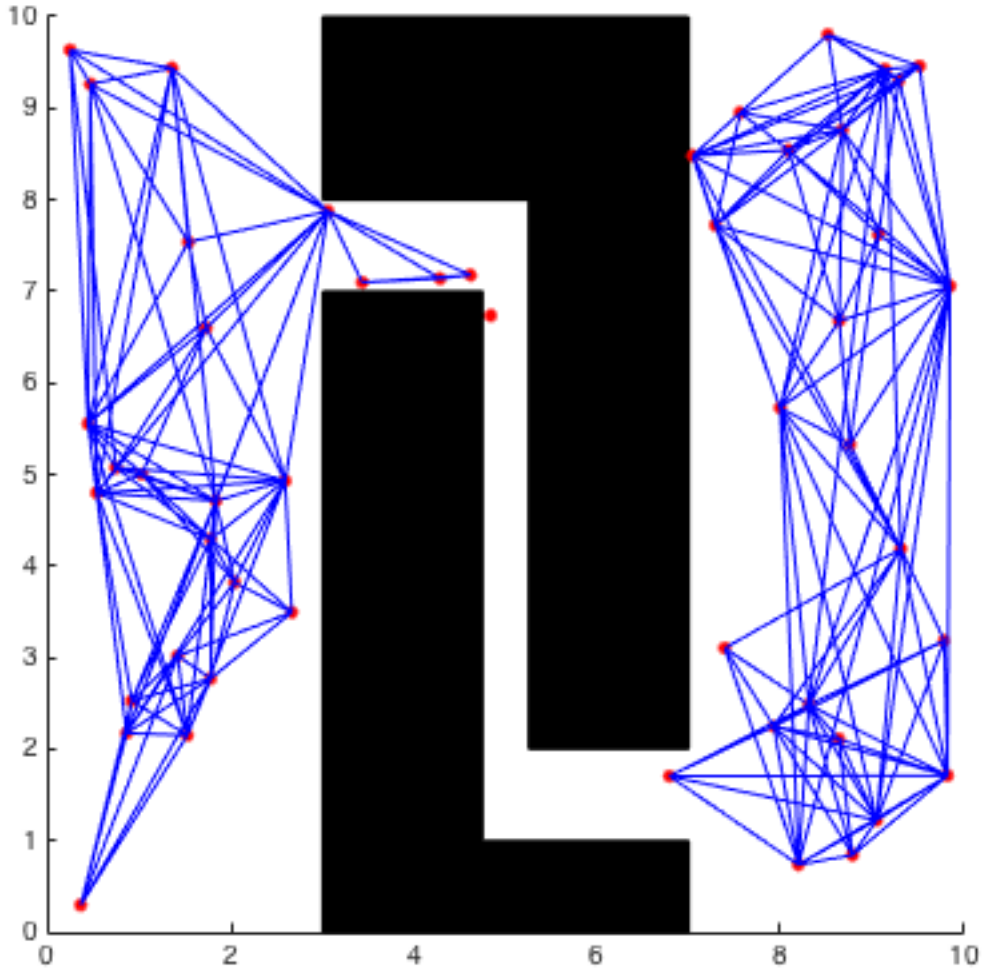
# SECTION 3.2

Penn

- There are a couple of characteristics of Random Sampling Based approaches that are worth noting.
- First off, while these methods work very well in practice they are not strictly speaking complete.
- A complete path planning algorithm would find a path if one existed and report failure if it didn't.

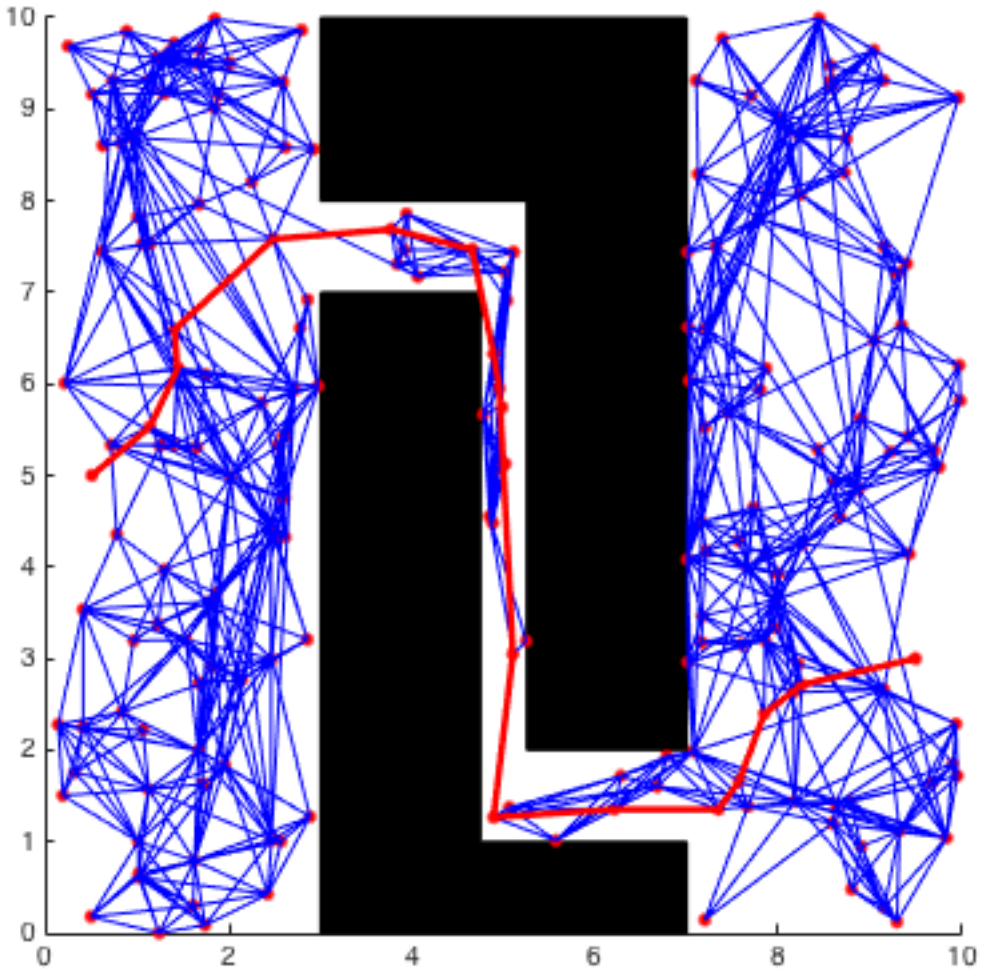
- **With the PRM procedure it is possible to have a situation where the algorithm would fail to find a path even when one exists if the sampling procedure fails to generate an appropriate set of samples.**

# Twisty Passageway – Failure Case




Penn

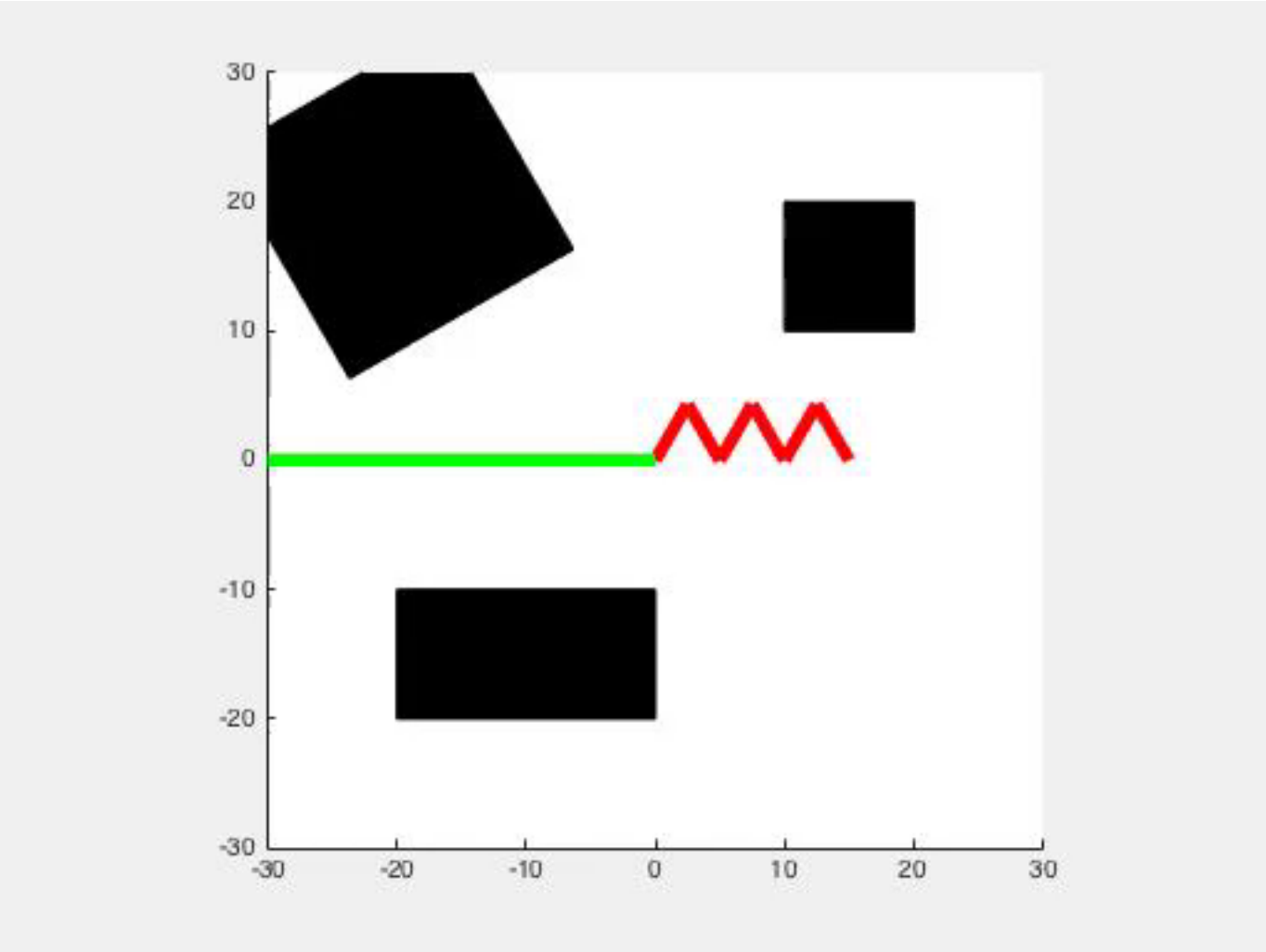
# Twisty Passageway – Success via denser samples



- **What we can say is that if there is a route and the planner keeps adding random samples it will, eventually find a solution.**
- **However it may take a long time to generate a sufficient number of samples.**

- **A real advantage of these PRM based planners is that they can be applied to systems with lots of degrees of freedom as opposed to grid based sampling schemes which are typically restricted to problems in 2 or 3 dimensions.**

# Trajectory for 6 degree of freedom arm



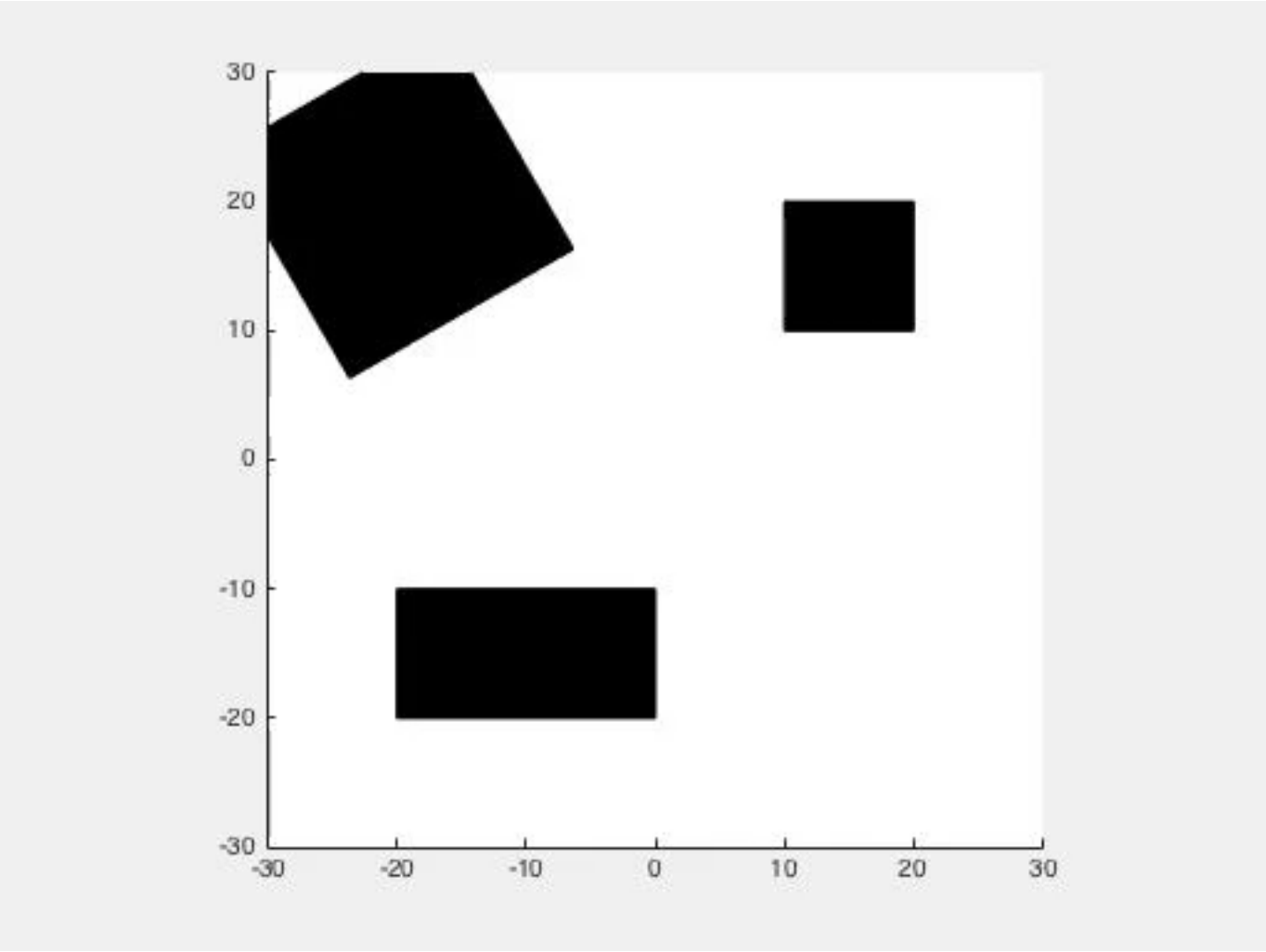
- In conclusion by relaxing the notion of completeness a bit and embracing the power of randomization these probablistic road map algorithms provide effective methods for planning routes that can be applied to a wide range of robotic systems including systems with many degrees of freedom.

Rapidly Exploring Random Trees

# SECTION 3.3

Penn

# Example of RRT growth in 2D Space



# RRT Procedure

- Add start node to tree
- Repeat n times
  - Generate a random configuration, x
  - If x is in freespace using the CollisionCheck function

    Find y, the closest node in the tree to the random configuration

    If (Dist (x, y) > delta) – Check if x is too far from y

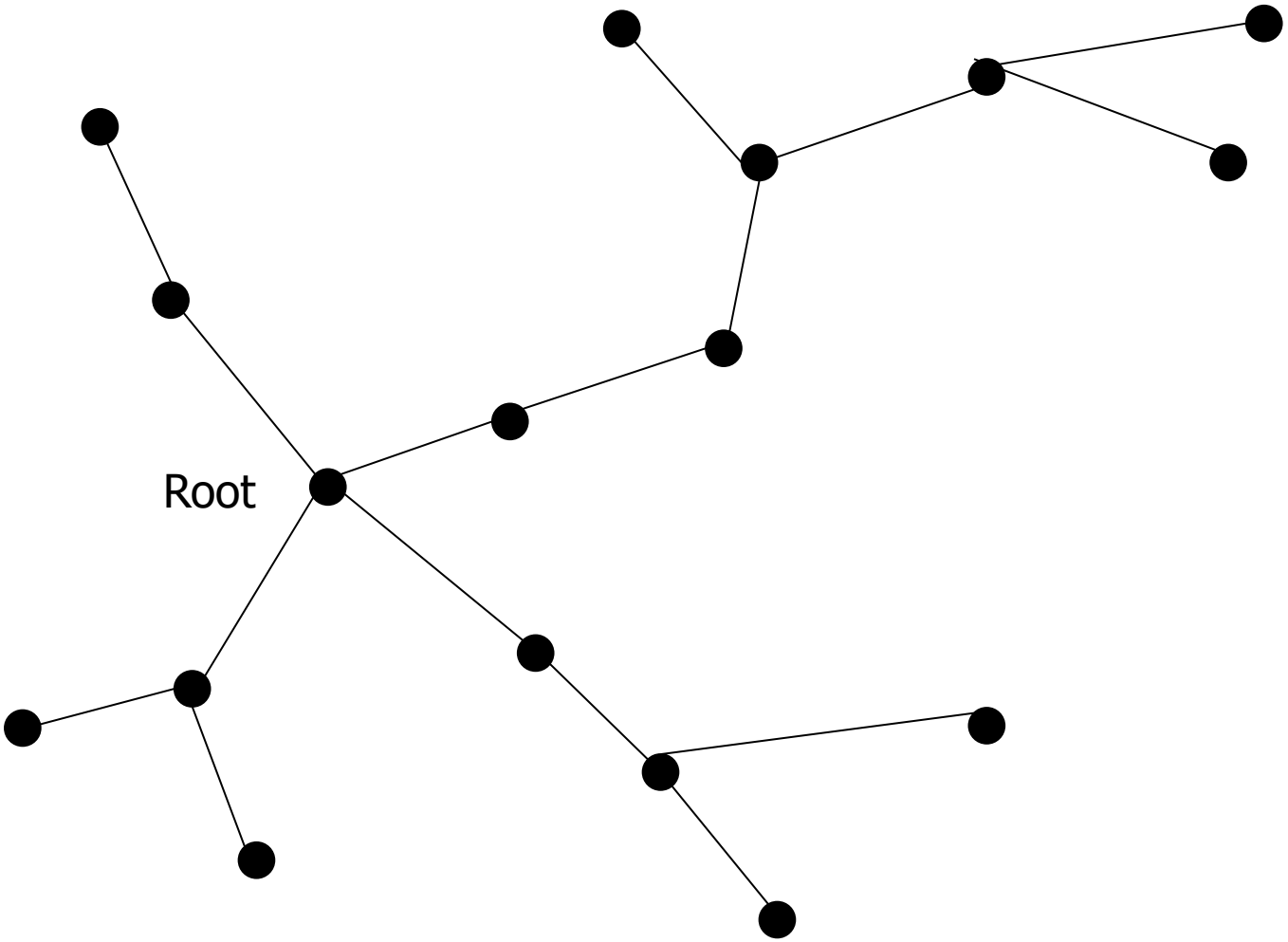
      Find a configuration, z, that is along the path from x to y such that Dist(z,y) <= delta

      x = z;

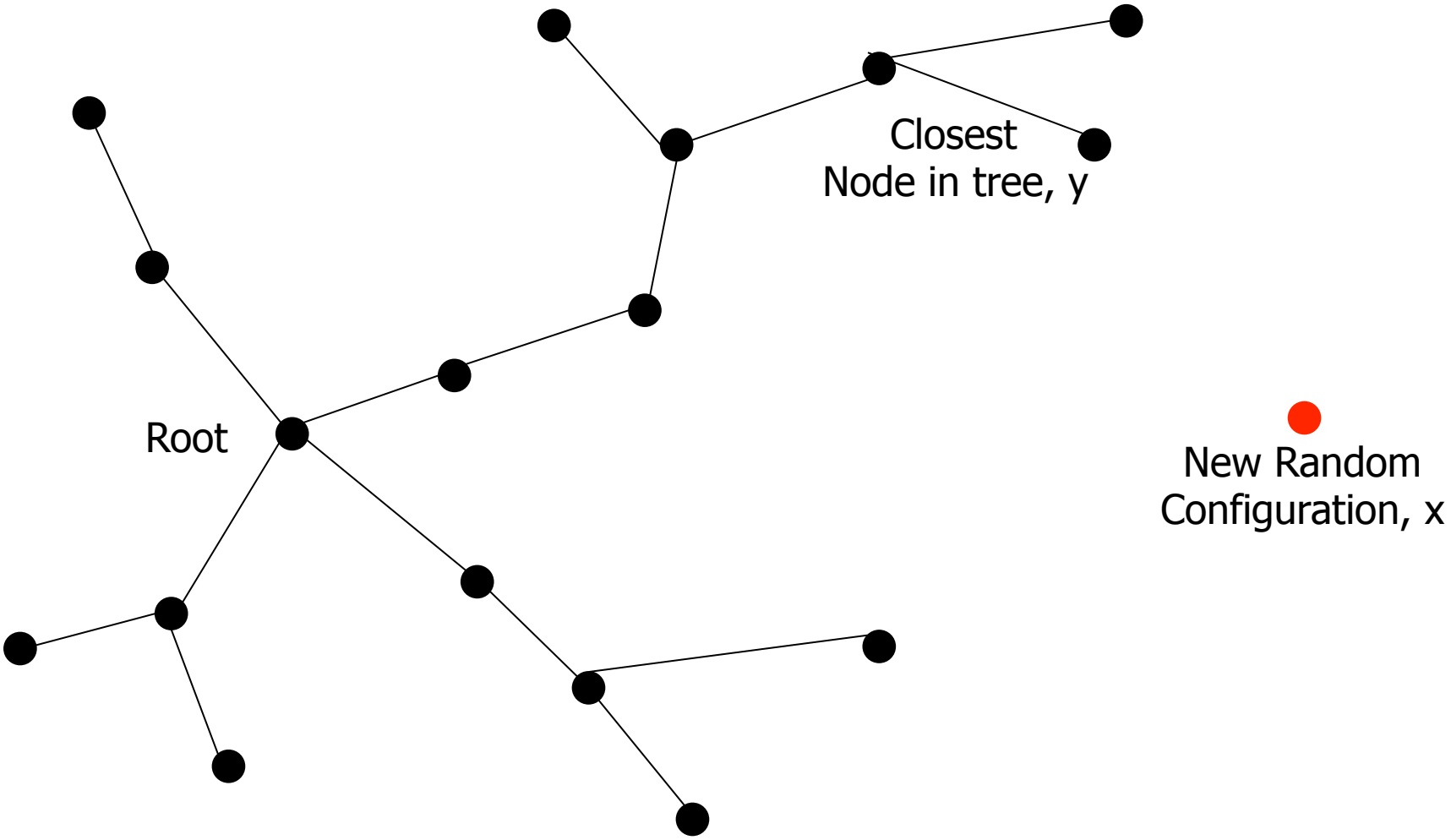
    If (LocalPlanner (x,y)) – Check if you can get from x to y

      Add x to the tree with y as its parent

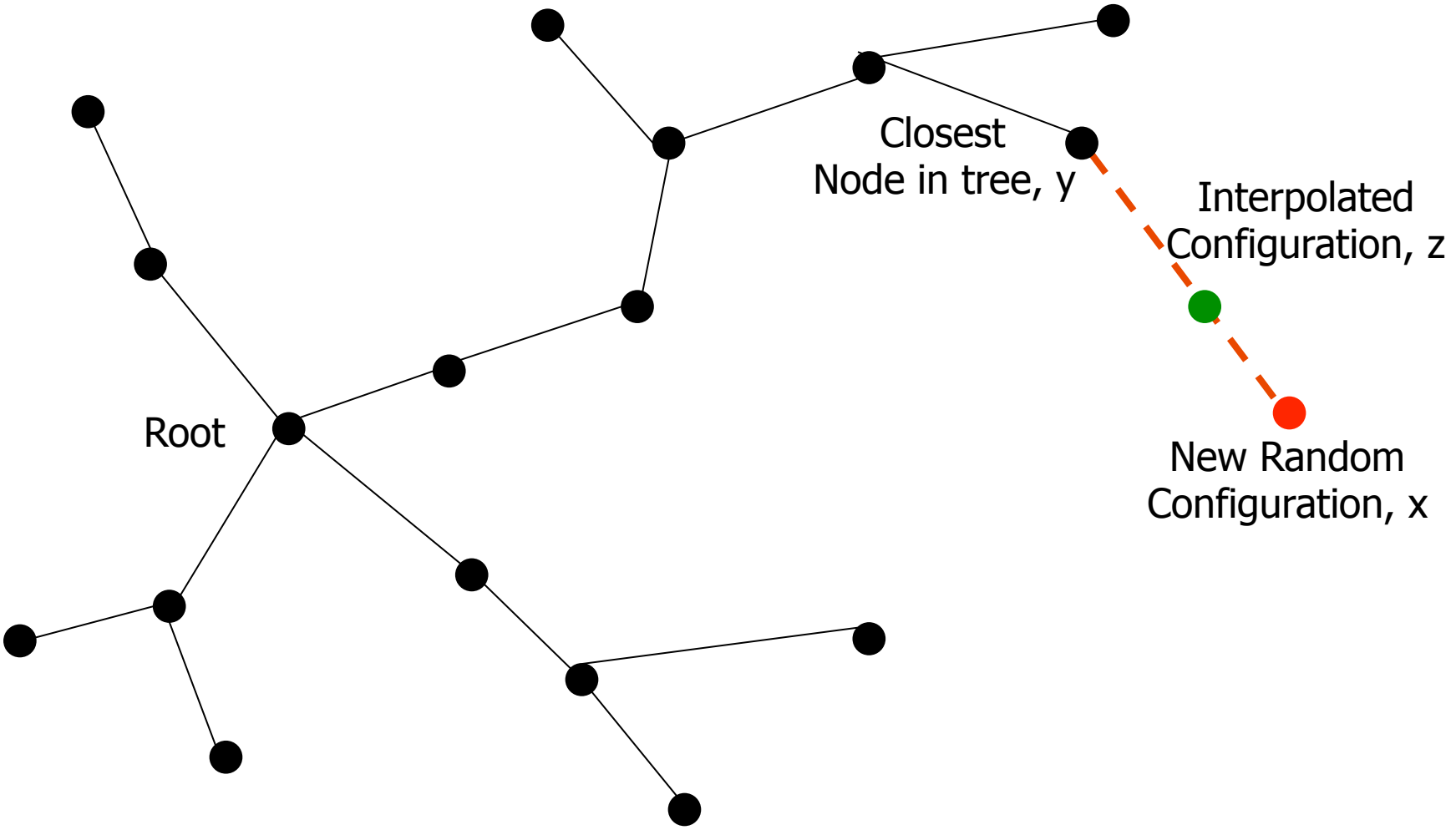
# RRT Extension Procedure – Initial tree



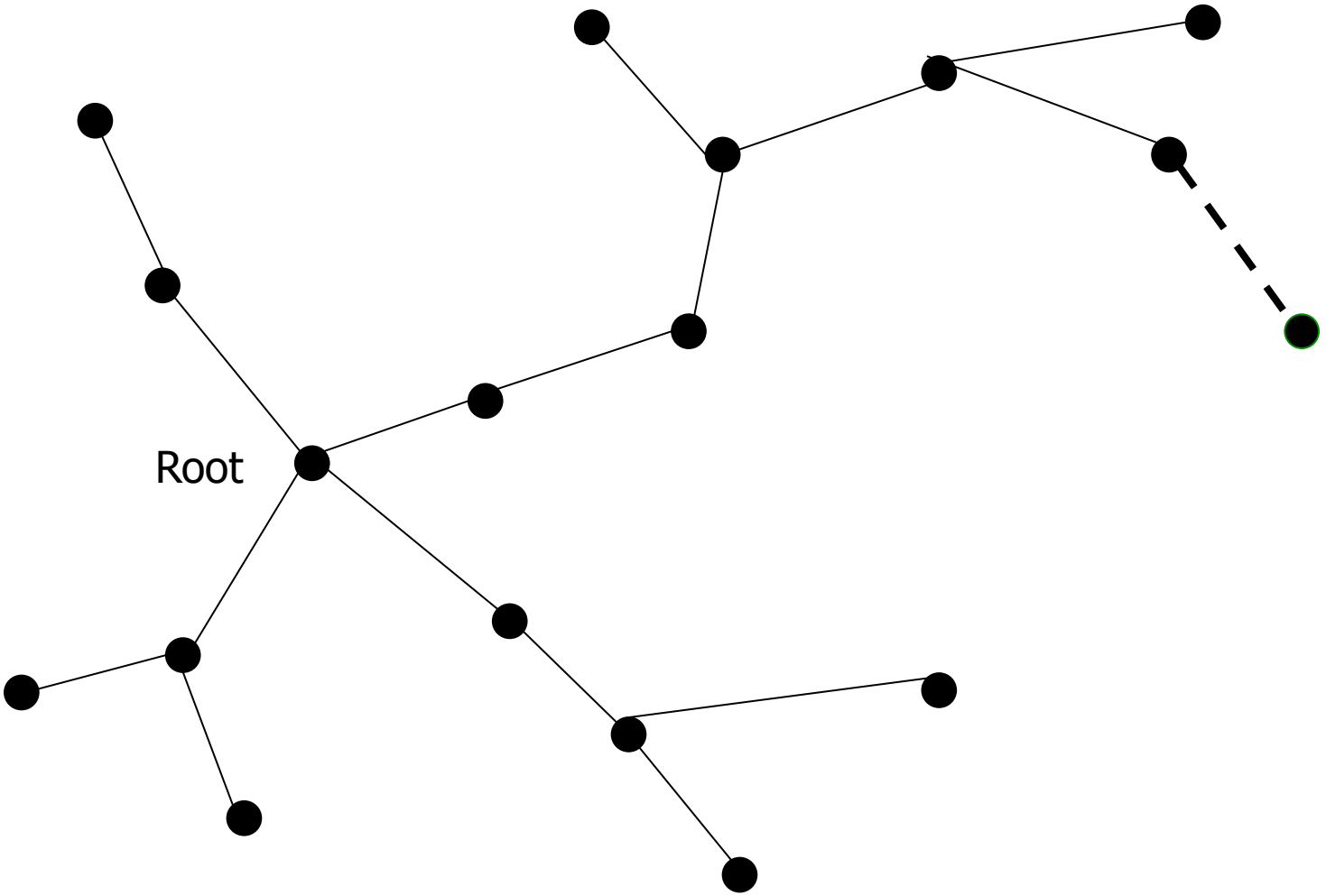
# RRT Extension Procedure



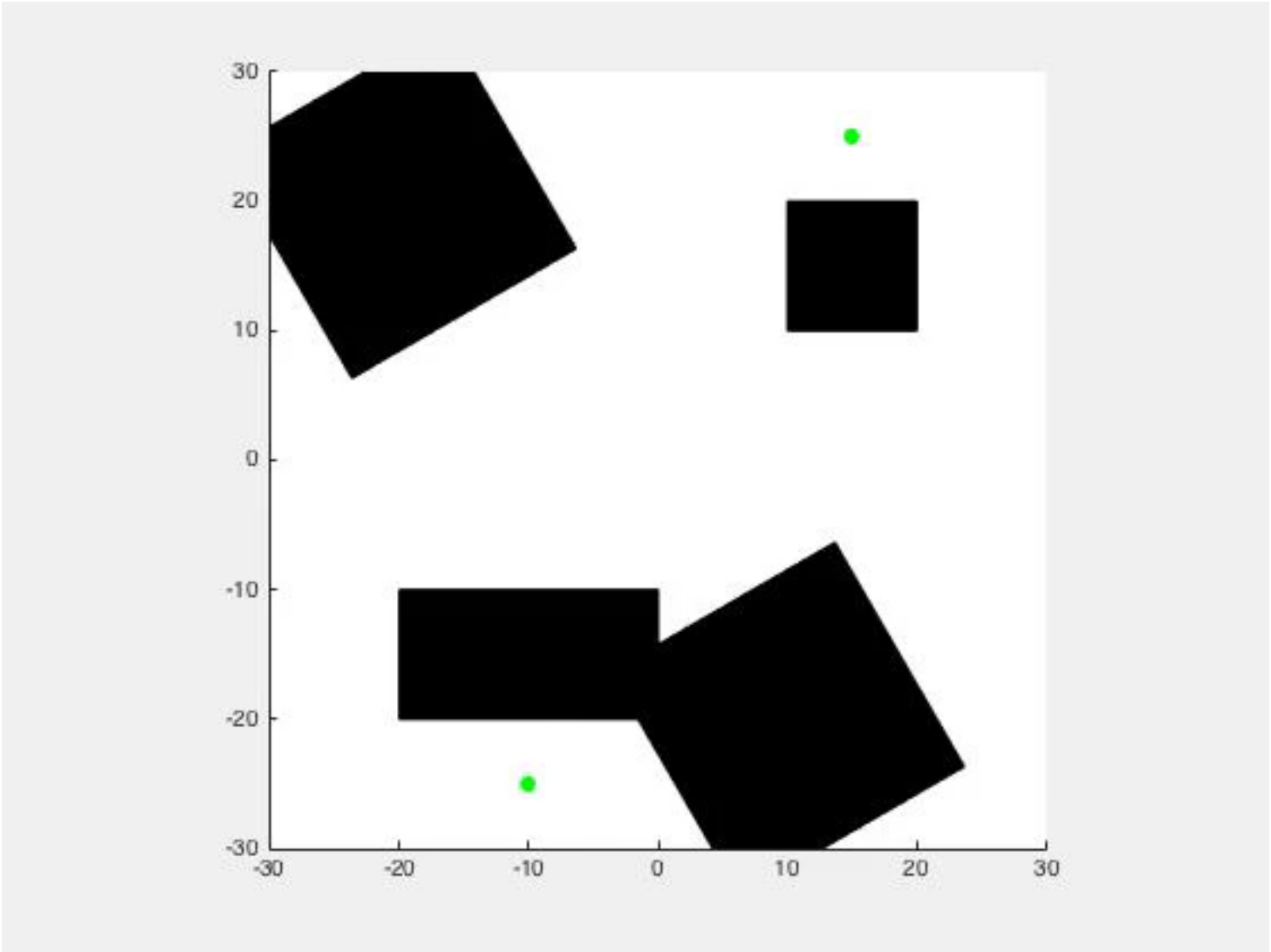
# RRT Extension Procedure



# RRT Extension Procedure – Graph after extension



# RRT Planning using 2 trees



# RRT 2 tree procedure

- While not done
  - Extend Tree A by adding a new node, x
  - Find the closest node in Tree B to x, y
  - If (LocalPlanner(x,y)) – Check if you can bridge the 2 trees

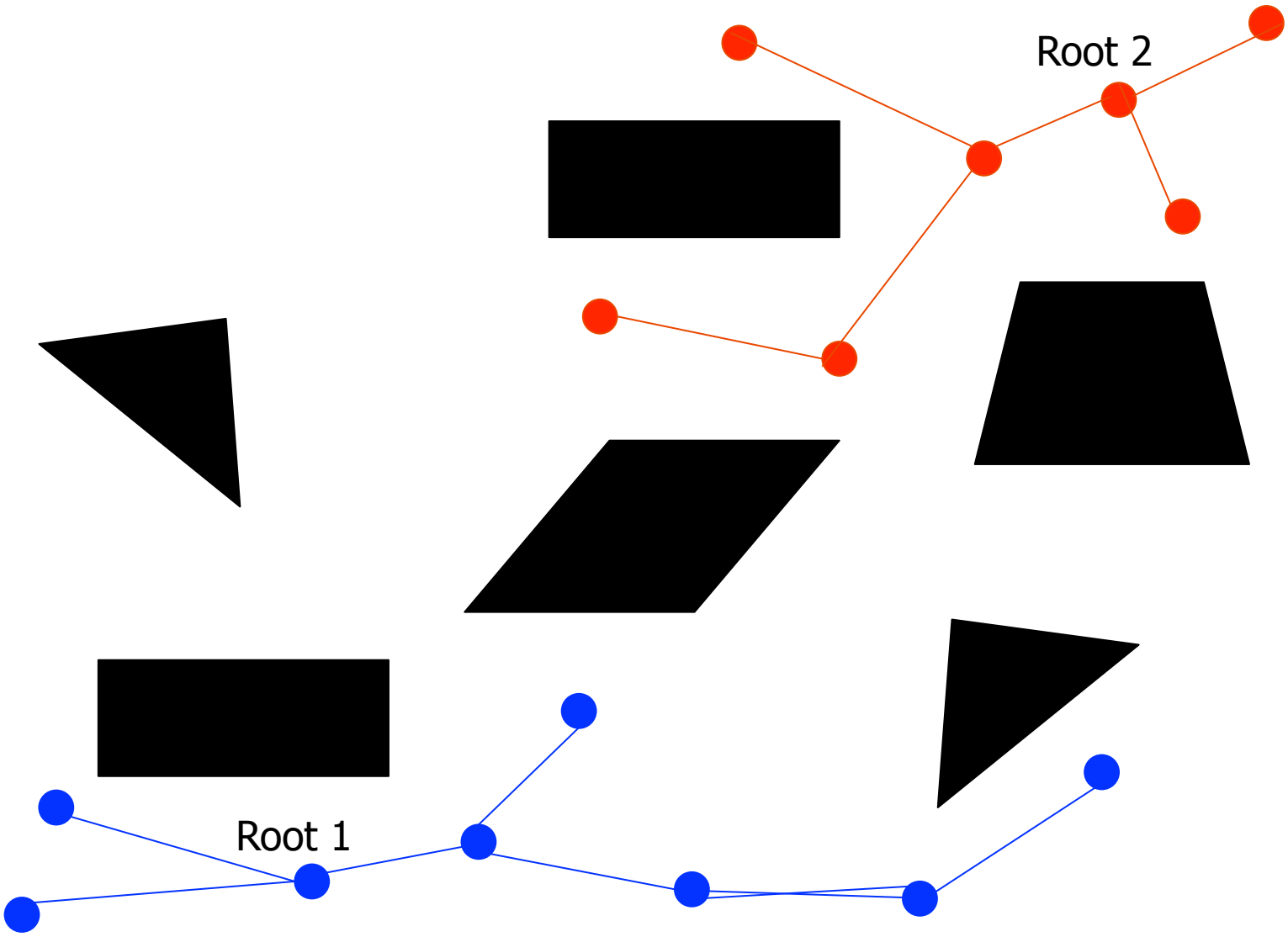
    Add edge between x and y.

    This completes a route between the root of Tree A and the root of Tree B. Return this route

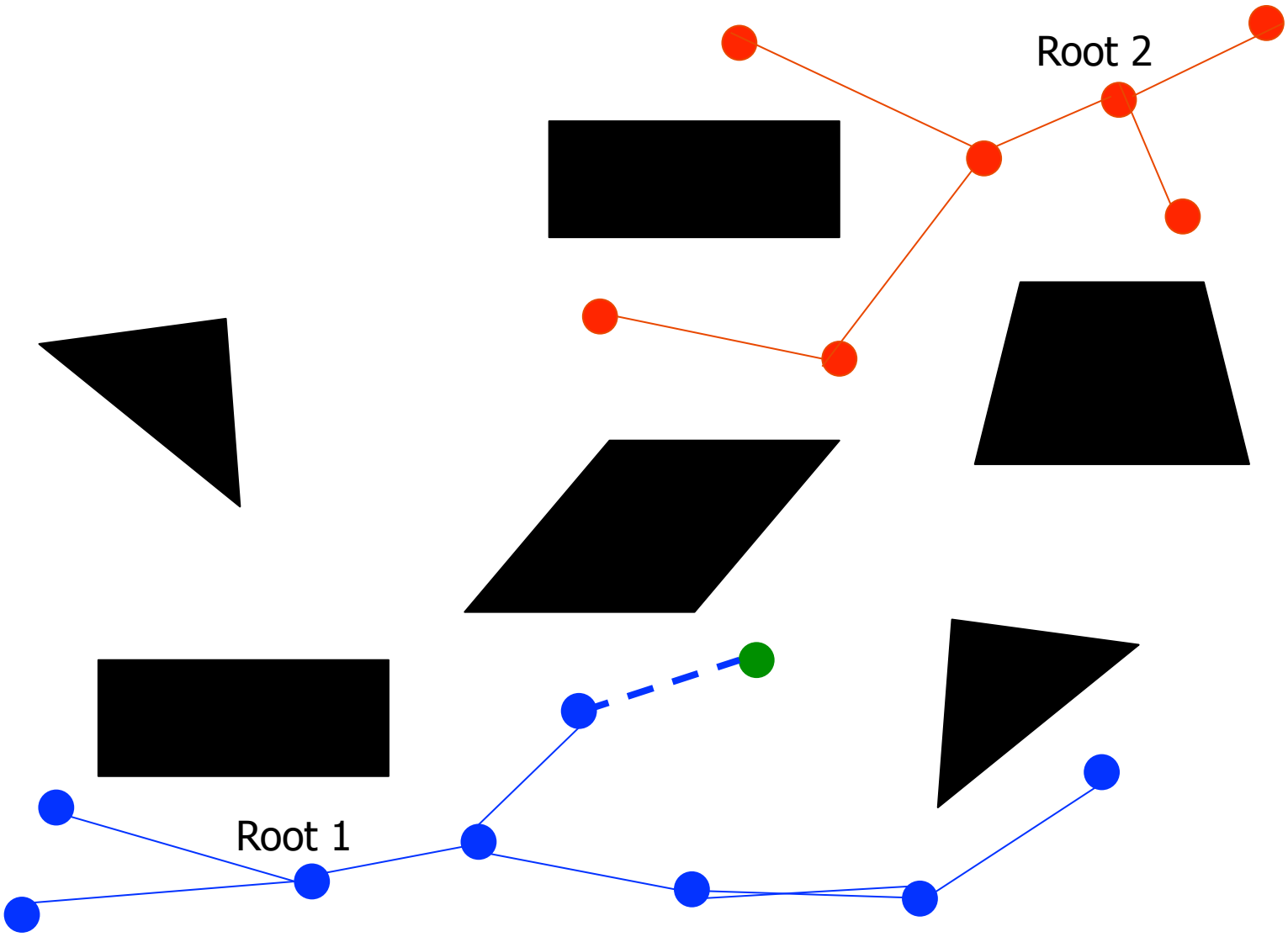
    Else

    Swap Tree A and Tree B

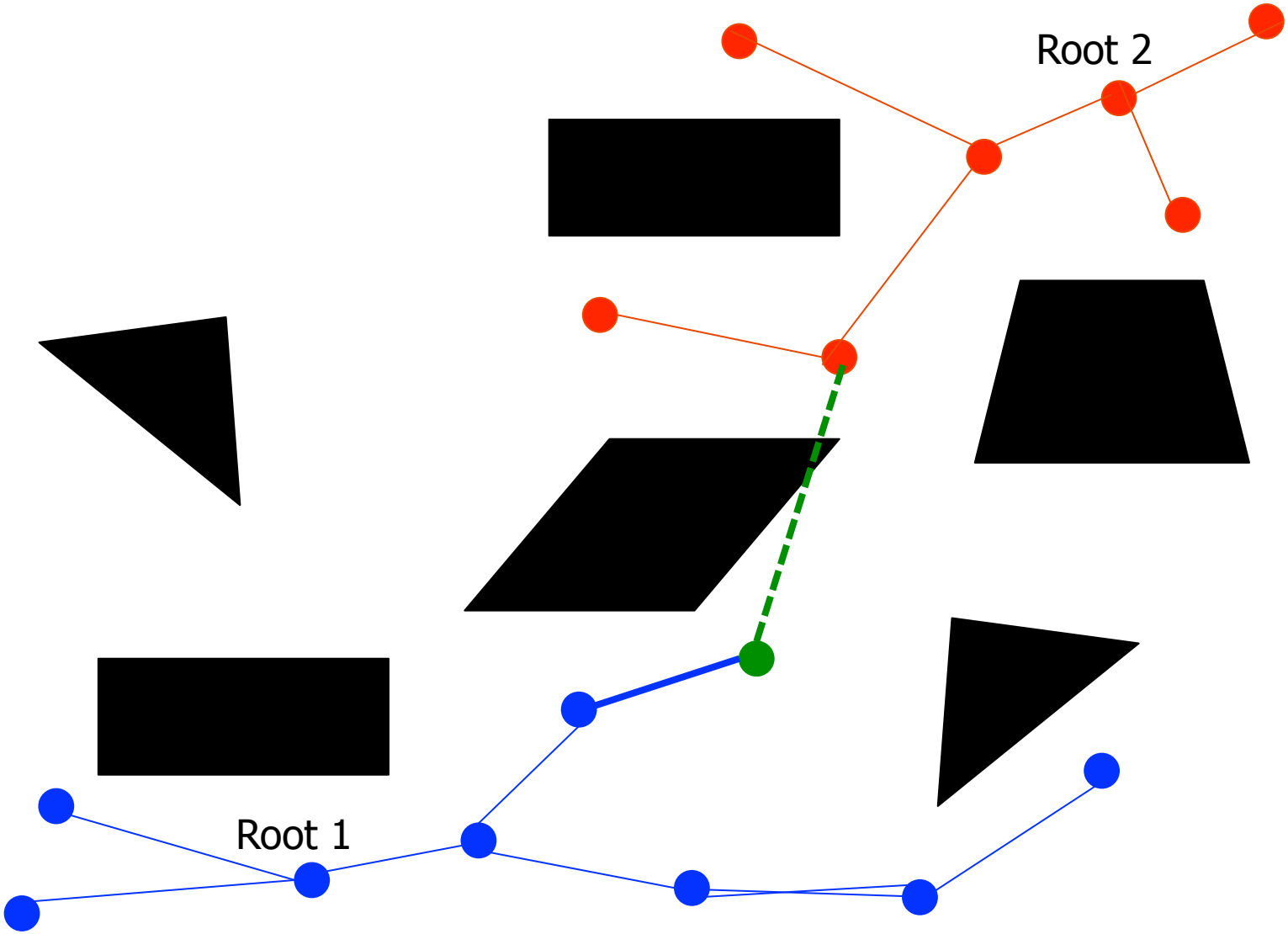
# RRT 2 Tree Procedure



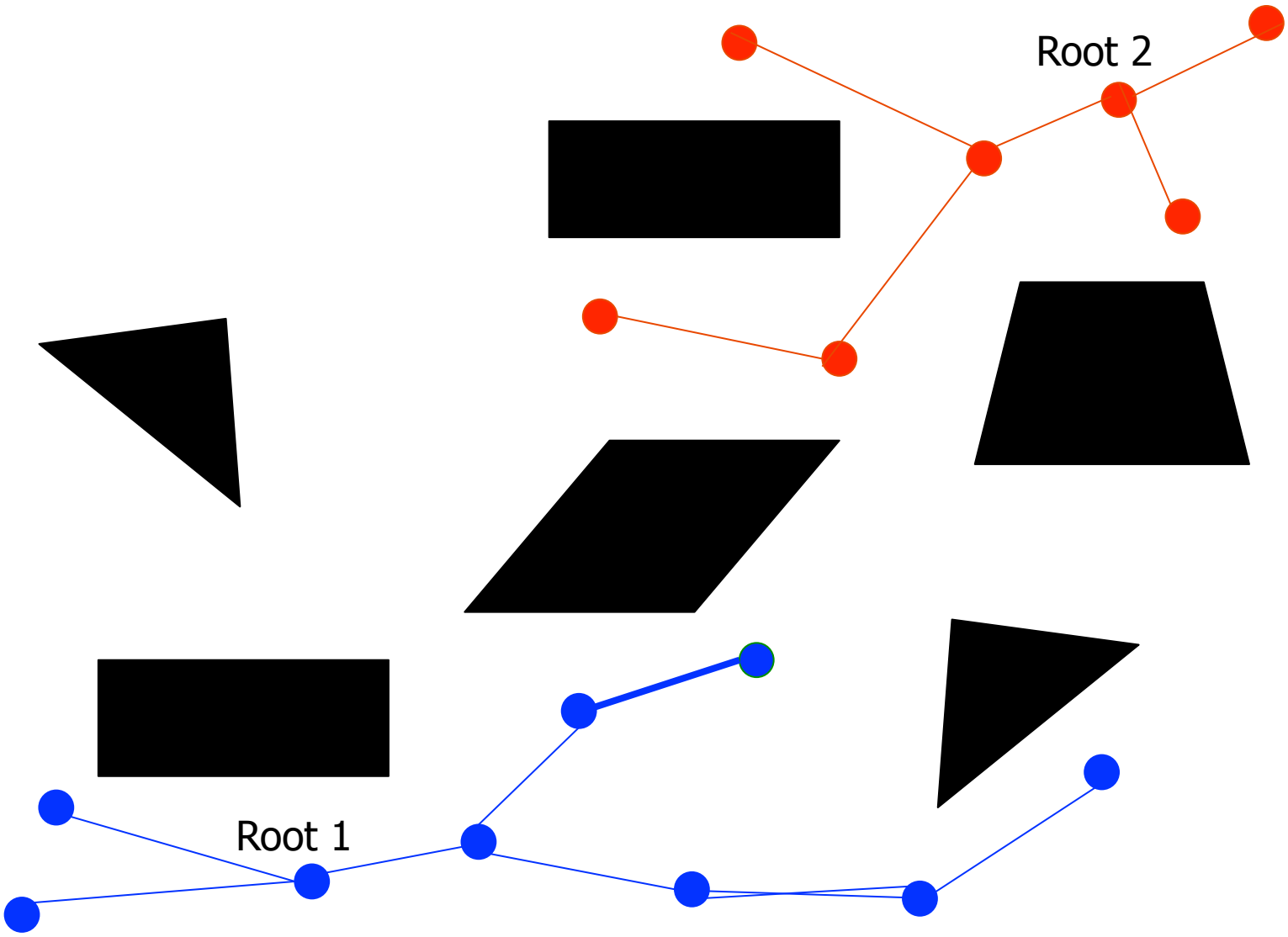
# RRT 2 Tree Procedure



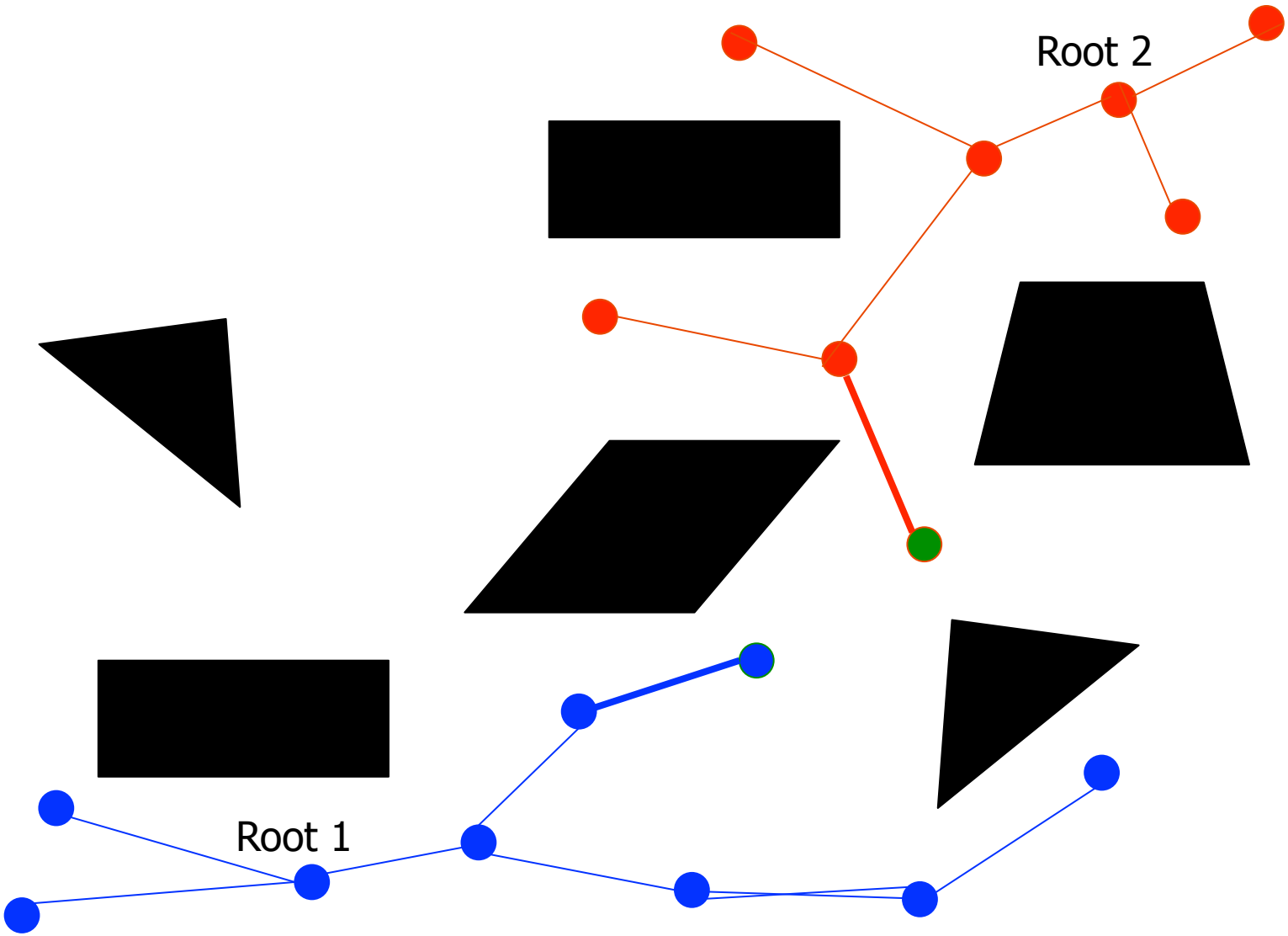
# RRT 2 Tree Procedure



# RRT 2 Tree Procedure



# RRT 2 Tree Procedure



# RRT 2 Tree Procedure